Amendments to the Claims:

1. (previously presented) The use of multiple threads in association with a network processor and accessible data available in a tree search structure, including the steps of:
   a) providing multiple instruction execution threads as independent processes in a sequential time frame;
   b) queuing the multiple execution threads to have overlapping access to the accessible data available in said tree search structure;
   c) executing a first thread in the queue; and
   d) transferring control of the execution to the next thread in the queue upon the occurrence of an event that causes execution of the first thread to stall.

2. (original) The use of the multiple threads according to claim 1 wherein the control of the execution is temporarily transferred to the next thread when execution stalls due to a short latency event, and the control is returned to the original thread when the event is completed.

3. (original) The use of multiple threads according to claim 2 wherein a processor instruction is encoded to select a short latency event.

4. (original) The use of the multiple threads according to claim 1 wherein full control of the execution is transferred to the next thread when execution of the first thread stalls due to a long latency event.

5. (original) The use of multiple threads according to claim 4 wherein a processor instruction is encoded to select a long latency event.

6. (previously presented) The use of multiple threads according to claim 1 including queuing the threads to provide rapid distribution of access to shared memory.

7. (original) The use of the multiple threads according to claim 1 wherein the threads have overlapping access to shared remote storage via a pipelined coprocessor by operating within different phases of a pipeline of the coprocessor.

8. (original) The use of the multiple threads according to claim 1 further including the step of providing a separate instruction pre-fetch buffer for each execution thread, and collecting instructions in a prefetch buffer for its execution thread when the thread is idle and when the instruction bandwidth is not being fully utilized.

9. (original) The use of the multiple threads according to claim 1 wherein the threads are used with zero overhead to switch execution from one thread to the next.

10. (original) The use of the multiple threads according to claim 9 wherein each thread is given access to general purpose registers and local data storage to enable switching with zero overhead.

11. (previously presented) A network processor that uses multiple threads to access data, including:

a) a CPU configured with multiple instruction execution threads as independent processes in a sequential time frame;

b) a thread execution control for

1) queuing the multiple execution threads to have overlapping access to the accessible data;

2) executing a first thread in the queue; and

3) transferring control of the execution to the next thread in the queue upon the occurrence of an event that causes execution of the first thread to stall.

12. (original) A processing system utilizing multiple threads according to claim 11 wherein the thread execution control includes control logic for temporarily transferring the control to the next thread when execution stalls due to a short latency event, and for returning control to the original thread when the latency event is completed.

13. (previously presented) The processing system according to claim 12 wherein a processor instruction is encoded to select a short latency event.

14. (original) The processing system according to claim 11 wherein the control transfer means includes the means for transferring full control of the execution to the next thread when execution of the first thread stalls due to a long latency event.

15. (original) The processing system according to claim 14 wherein a processor instruction is encoded to select a long latency event.

16. (original) The processing system according to claim 11 including means to queue

the threads to provide rapid distribution of access to shared memory.

17. (original) The processing system according to claim 16 wherein the threads have overlapping access to shared remote storage via a pipelined coprocessor by operating within different phases of a pipeline of the coprocessor.

18. (original) The processing system according to claim 11 further including a separate instruction pre-fetch buffer for each execution thread, and means for collecting instructions in a prefetch buffer for an idle execution thread when the instruction bandwidth is not being fully utilized.

19. (original) The system according to claim 11 wherein the processor uses zero overhead to switch execution from one thread to the next.

20. (original) The system according to claim 19 wherein each thread is given access to an array of general purpose registers and local data storage to enable switching with zero overhead.

21. (original) The system according to claim 20 wherein the general purpose registers and the local data storage are made available to the processor by providing one address bit under the control of the thread execution control logic and by providing the remaining address bits under the control of the processor.

22. (original) The system according to claim 20 wherein the processor is capable of simultaneously addressing multiple register arrays, and the thread execution control logic includes a selector to select which array will be delivered to the processor for

a given thread.

23. (original) The system according to claim 20 wherein the local data storage is fully addressable by the processor, an index register is contained within the register array, and the thread execution control has no address control over the local data storage or the register arrays.

24. (currently amended) A network processor configuration comprising:

a CPU with multiple threads;

an instruction memory, and a separate prefetch queue for each thread between the instruction memory and the CPU;

an array of general purposes registers, said array communicating with the CPU;

a local data storage including separate storage space for each thread;

a thread execution control for the general register array and the local data storage;

a first coprocessor connecting the CPU to a local data storage, said thread execution control including a priority FIFO buffer to store thread numbers;

a shared remote storage; and

a pipelined coprocessor connecting the shared remote storage and the CPU.

25. (previously presented) The processor configuration according to claim 24 wherein the thread execution control further includes a plurality of thread control state machines, one for each execution thread, and an arbiter responsive to signals from the FIFO buffer and the state machine to determine thread execution priority.

26. (original) The use of prefetch buffers in connection with a plurality of independent instruction threads used to process data in a Network Processor comprising the steps of:

a) associating each thread with a prefetch buffer;

b) determining whether a buffer associated with an execution thread is full;

c) determining whether the thread associated with the buffer is active; and

d) during periods that the buffer is not being used by an active execution thread, enabling the buffer to prefetch instructions for the execution thread.

27. (previously presented) A thread execution control useful for the efficient execution of independent threads comprising:

a) a priority FIFO buffer for storing thread numbers;

b) a plurality of thread control state machines, one for each thread; and

c) an arbiter for determining the thread execution priority among multiple threads based upon signals outputted from the FIFO buffer and the state machines.

28. (currently amended) The thread execution control according to claim 27 wherein the FIFO includes :

a) means for loading a thread number into the FIFO when a packet is dispatched to the processor;

b) means for unloading a thread number from the FIFO when a packet

has been enqueued for transmission;

(c) thread number transfer from highest priority to lowest priority in the FIFO when a long latency event occurs, and

(d) ~~the~~ thread outlets of the FIFO used to determine priority depending on the length of time a thread has been in FIFO.

29. (original) The thread execution control according to claim 27 wherein the arbiter controls the priority of execution of multiple independent threads based on the Boolean expression:

$$G_n = R_n \cdot \{(P_A = n) + \overline{R_{PA}} \cdot (P_B = n) + \overline{R_{PA}} \cdot \overline{R_{PB}} \cdot (P_C = n) \cdots\}$$

where: G is a grant

R_n is a request from a given thread;

P_A, P_B and P_C represent threads ranked by alphabetical subscript according to priority;

$_n$ is a subscript identifying a thread by the bit or binary number

comprising

a) determining whether a request R is active or inactive;

b) determining the priority of the threads;

c) matching the request R with the corresponding thread P; and

d) granting a request for execution if the request is active and if the corresponding thread P has the highest priority.

30. (currently amended) The thread execution control according to claim 27 wherein the thread control state machine comprises control logic to :

(a) dispatch a packet to a thread;

(b) move the thread from an initialize state to a ready state;

(c) request execution cycles for the thread;

(d) move the thread to the execute state upon grant by the arbiter of an execution cycle;

(e) continue to request execution cycles while the thread is queued in the execute state; and

(f) return the thread to the initialize state if there is no latency event, or send the ~~packet~~ thread to the wait state upon occurrence of a latency event.

31. (previously presented) The thread executing control according to claim 28 wherein the FIFO further includes means to detect occurrence of latency events.